2

NASA Contractor Report 187499

ICASE Report No. 91-5

ICASE

MULTIPHASE COMPLETE EXCHANGE ON A CIRCUIT SWITCHED HYPERCUBE

Shahid H. Bokhari

Contract No. NAS1-18605
January 1991

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665-5225

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Multiphase Complete Exchange on a Circuit Switched Hypercube[*]

Shahid H. Bokhari

Department of Electrical Engineering

University of Engineering & Technology, Lahore, Pakistan

and

ICASE, NASA Langley Research Center

Hampton, Virginia

Abstract

On a distributed memory parallel computer, the complete exchange (all-to-all personalized) communication pattern requires each of n processors to send a different block of data to each of the remaining $n-1$ processors. This pattern is at the heart of many important algorithms, most notably the matrix transpose.

For a circuit switched hypercube of dimension d ($n = 2^d$), two algorithms for achieving complete exchange are known. These are (1) the Standard Exchange approach that employs d transmissions of size 2^{d-1} blocks each and is useful for small block sizes, and (2) the Optimal Circuit Switched algorithm that employs $2^d - 1$ transmissions of 1 block each and is best for large block sizes.

A unified multiphase algorithm is described that includes these two algorithms as special cases. The complete exchange on a hypercube of dimension d and block size m is achieved by carrying out k partial exchanges on subcubes of dimension $d_i, \Sigma_{i=1}^{k} d_i = d$ and effective block size $m_i = m2^{d-d_i}$. When $k = d$ and all $d_i = 1$, this corresponds to algorithm (1) above. For the case of $k = 1$ and $d_1 = d$, this becomes the circuit switched algorithm (2). Changing the subcube dimensions d_i varies the effective block size and permits a compromise between the data permutation and block transmission overhead of (1) and the startup overhead of (2).

For a hypercube of dimension d, the number of possible combinations of subcubes is $p(d)$, the number of partitions of the integer d. This is an exponential but very slowly growing function (e.g. $p(7) = 15, p(10) = 42$) and it is feasible to enumerate over these partitions to discover the best combination for a given message size.

This approach has been analyzed for, and implemented on, the Intel iPSC-860 circuit switched hypercube. Measurements show good agreement with predictions and demonstrate that the multiphase approach can substantially improve performance for block sizes in the 0–160 byte range. This range, which corresponds to 0–40 floating point numbers per processor, is commonly encountered in practical numeric applications. The multiphase technique is applicable to all circuit-switched hypercubes that use the common 'e-cube' routing strategy.

[*]Research supported by the National Aeronautics and Space Administration under NASA contract NAS1-18605 while the author was in residence at the Institute for Computer Applications in Science & Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225.

1 Introduction

On a distributed memory parallel computer system, the complete exchange communication pattern requires each of n processors to send a different m byte block of data to each of the remaining $n-1$ processors. This communication pattern arises in many important applications such as matrix transpose, matrix-vector multiply, 2-dimensional FFTs, distributed table look-ups etc. It is also important in its own right since, being equivalent to a complete directed graph, it is the densest communication requirement that can be imposed on an interconnection network. The time required to carry out the complete exchange is an important measure of the power of a distributed memory parallel computer system.

There are two algorithms for complete exchange on circuit switched hypercubes like the Intel iPSC-2, Intel iPSC-860, and Ncube-2. The first is the Standard Exchange algorithm: for an d-dimensional hypercube, this algorithm uses d transmissions of 2^{d-1} blocks each. On circuit switched machines this algorithm is useful for small block sizes ($< \approx 200$ bytes). The second is the Optimal Circuit Switched algorithm, that uses $2^d - 1$ transmissions of 1 block each: the transmissions are carefully scheduled to avoid link contention.

We describe in this paper a unified multiphase algorithm that carries out the complete exchange on a hypercube of dimension d as set of k "partial" exchanges on subcubes of dimensions d_i, $\Sigma_{i=1}^{k} d_i = d$ with effective block size $m_i = m2^{d-d_i}$. The motivation here is to reduce the time required for the complete exchange by compromising between the data permutation and block transmission overhead of the Standard Exchange algorithm and the startup overhead of the Optimal algorithm.

For a hypercube of dimension d there are $p(d)$ possible generalized algorithms, where $p(d)$ is the number of partitions of the integer d. Although $p(d)$ is an exponential function, it grows very slowly. For example, $p(7) = 15$, $p(10) = 42$, and $p(20) = 672$. It is thus quite feasible to enumerate over all partitions to find the algorithm best suited for a given block size.

For the case where $k = d$ and each $d_i = 1$, the unified algorithm degenerates into the Standard Exchange algorithm. When $k = 1$ and $d_1 = d$, it becomes the Optimal algorithm. The unified algorithm thus includes the two known algorithms as special (although extreme) cases.

Measurements on the Intel iPSC-860 hypercube show that the multiphase approach can substantially improve performance for block sizes in the 0–160 byte range. This range corresponds to 0–40 floating point numbers and is commonly encountered in practical numeric applications. While our measurements are for the Intel iPSC-860, our techniques are applicable to all circuit switched hypercubes that use the common 'e-cube' routing strategy. The older Intel iPSC-2 and the Ncube-2 are examples of such machines.

In Section 2 of this paper we describe the essential features of circuit switched hypercubes. We discuss the complete exchange pattern in Section 3. Section 4 describes the two previously known algorithms for the complete exchange.

The major theoretical results of this paper are presented in Section 5. We introduce the unified multiphase algorithm with an example and then go on to describe partial exchanges. This Section concludes with a presentation of the general algorithm. In Section 6 we describe how an enumeration approach can be used to obtain the optimal set of subcube dimensions. Details of implementation on the iPSC-860 are given in Section 7. In Section 8 we discuss our observed timings and compare them with predictions. We conclude with a discussion of our results and projections for future research in Section 9.

2 Circuit Switched Hypercubes

The interconnection network of a 32 node hypercube is shown in Figure 1. The labeled vertices hanging from each vertex of the network represent processors of the hypercube. Two processors in the network are connected if and only if the binary representations of their labels differ in exactly one bit. Circuit-switched communications differentiate the newer hypercubes, such as the Intel iPSC-2 and iPSC-860, and the Ncube-2 from older machines. In these machines, a dedicated path is set up between two processors when communication is desired. Messages then flow through this path without involving intervening processors. The path between source and destination is determined by the 'e-cube' routing algorithm: starting with the right hand side of the binary label of the source processor, we move to the processor whose label more closely matches the label of the destination processor. This process is repeated until the destination is reached.

The user has no control over *how* a message is routed between two processors. The fixed routing algorithm completely determines this path. Because of this, we can encounter *edge* and *node* contention. Edge contention is the sharing of an edge (i.e. a communication link) by two or more paths. Similarly, node contention is the sharing of a node.

Figure 1 illustrates paths from 0 to 31 (solid), 2 to 23 (dashed) and 14 to 11 (dotted). The *lengths* of these paths (the *distance* between source and destination) are 5, 3 and 2 respectively. The paths $0 \rightarrow 31$ and $2 \rightarrow 23$ share the edge 3–7, while the paths $0 \rightarrow 31$ and $14 \rightarrow 11$ share node 15. Measurements on the iPSC-860 [2] reveal that edge contention has a disastrous impact on communication time, while node contention has no measurable effect.

The Intel iPSC-2 and IPSC-860 are among the first commercial examples of circuit-switched machines. Since circuit switching provides very fast communications, it is generally felt that it eliminates most, if not all, of the inefficiencies caused by communication overhead. In particular, it is a common belief that programmers can ignore the details of the interconnection network, since communication overhead is negligible. This is a mistaken belief since, as we shall see later in this paper, very careful consideration of the interconnection network is necessary if the full power of the machine is to be utilized.

3 The Complete Exchange Pattern

The Complete exchange communication pattern requires each of n processors of a parallel machine to send a different block of data to each of the remaining $n-1$ machines. This pattern arises when transposing a matrix of size $n \times n$ blocks that is mapped onto an n processor system (Figure 2). As shown in the bottom part of this figure, the transpose requires each processor to send 1 block to each of the remaining $n-1$ processors. The resulting communication pattern is a complete directed graph of n nodes.

The specific mapping of an $n \times n$ matrix onto an n processor system shown in Figure 2 is required when using the Alternating Directions Implicit (ADI) method for solving partial differential equations [5, 10]. This method requires access to the matrix by rows and by columns in successive phases, necessitating the heavy use of a transpose procedure. Similar requirements arise in matrix-matrix and matrix-vector multiplication, when the matrices

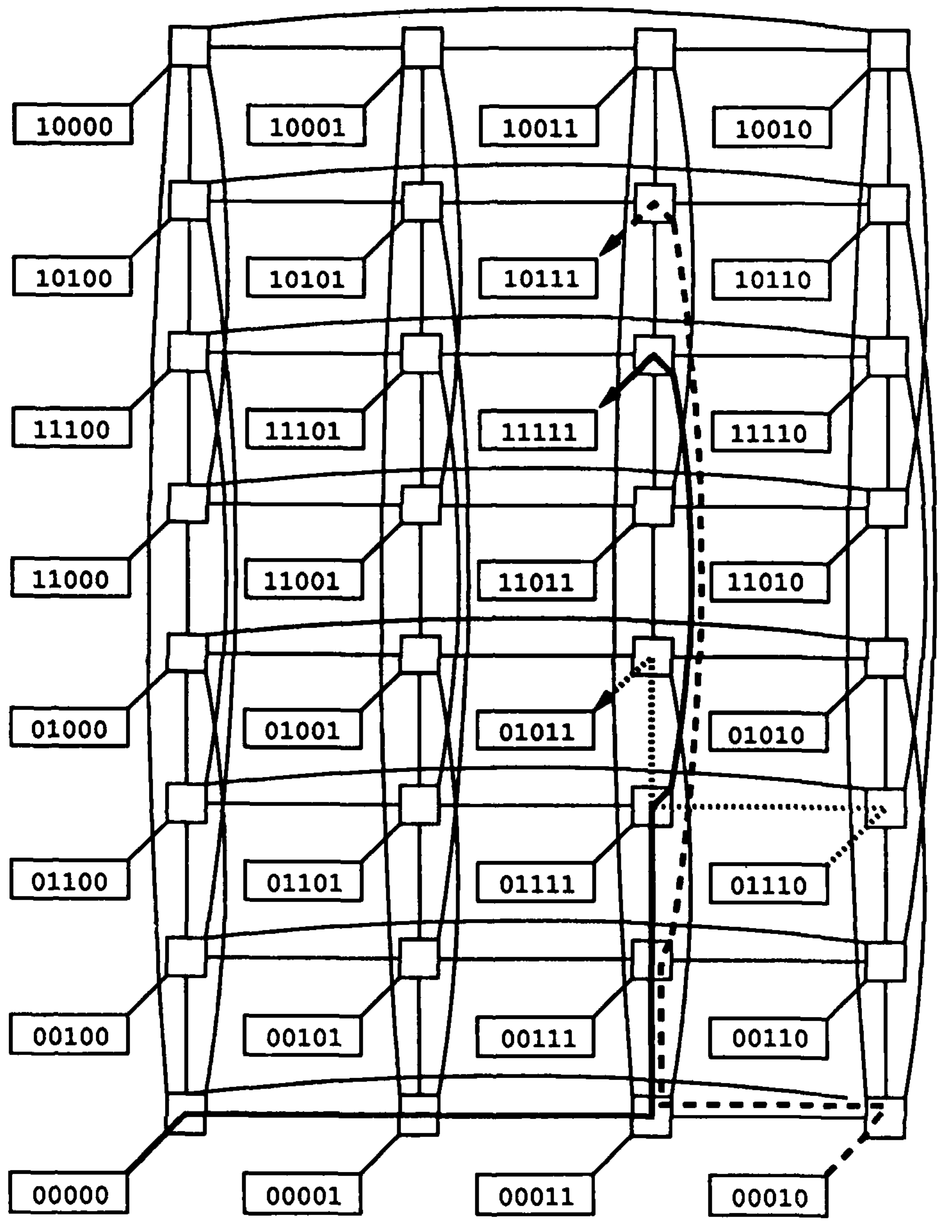

Figure 1: Interconnection network of a 32 node hypercube.

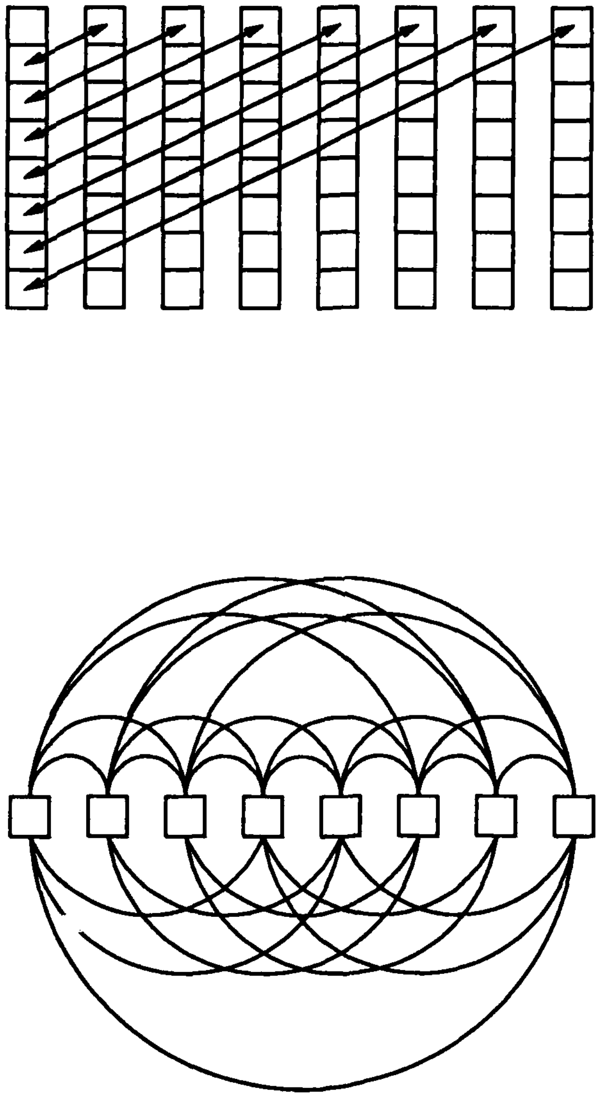

Figure 2: Matrix transpose/Complete exchange communication pattern

are mapped as described above.

The complete exchange pattern also arises in certain implementations of the 2-D FFT [11] and in distributed table lookup[12]. Being equivalent to a complete directed graph of n nodes, this pattern is of interest in its own right, since it is the densest communication requirement that can be imposed on an interconnection network. The time required to execute the complete exchange pattern is an upper bound for the time required by any pattern (which must necessarily be a subset of the complete directed graph).

Because of its widespread applications, it is worthwhile to investigate the time required to execute this pattern and to develop fast procedures for it, as we proceed to do in the following Sections.

4 Algorithms for Complete Exchange

We shall now discuss the two algorithms for complete exchange that are currently in use. Of these, Standard Exchange[7] is well known, while the Optimal Circuit Switched algorithm[13, 15] is a recent development. The former requires only $\log n$ transmissions of $n/2$ blocks each and has better performance for small block sizes. The latter uses $n-1$ transmissions of 1 block each and has better performance for large block sizes. Both algorithms completely avoid edge contention. The Exchange algorithm does this by communicating over unit distances. The Optimal algorithm avoids contention by using a carefully contrived schedule of transmissions.

4.1 The Standard Exchange Algorithm

The Standard Exchange algorithm [7] uses $\log n$ transmissions of size $n/2$ blocks each. All transmissions are along paths of length 1, thus there is no possibility of contention. This algorithm incurs overhead because of the shuffling of blocks and because it transmits $\frac{n}{2}\log n$ blocks instead of the optimal number, $n-1$. It is, nevertheless, competitive for small block sizes. This is because there are only $\log n$ transmissions (as opposed to $n-1$ for the algorithm discussed below) and thus the overhead of starting up a message is not incurred as frequently.

```
procedure Standard_Exchange;
begin
    for $j = d - 1$ downto 0 do
        begin
            if (bit $j$ of $mynumber = 0$) then
                message = blocks $n/2$ to $n - 1$
            else
                message = blocks 0 to $n/2 - 1$;
            send_message_to_processor($(mynumber) \oplus (2^j)$);
            shuffle blocks;
        end;
end;
```

4.2 The Optimal Circuit Switched Algorithm

The challenge in designing algorithms for circuit switched machines with fixed routing is to organize communications in such a way as to avoid or minimize edge contention. In the case of complete exchange, each processor must send its ith block to processor i, but is free to schedule its transmissions in order to avoid edge contention. There are many possible schedules that completely avoid contention. We will use the schedule developed by Schmiermund and Seidel[13]. This schedule has the property that the entire communication pattern is decomposed into a sequence of pairwise exchanges. This property is very useful when implementing complete exchanges on the Intel iPSC-2 and iPSC-860 because of certain idiosyncrasies of their communication hardware, as we shall see in Section 7. Other schedules are possible—some of these have advantages over certain ranges of block size. These are discussed further in [3].

```
procedure Optimal_Circuit_Switched;
begin
   for $i = 1$ to $n - 1$ do
      send_block_to_processor(($mynumber$) $\oplus$ ($i$));
end;
```

4.3 Analysis of Run Times

Let us define the following performance parameters of our hypercube.

	Description	Units
τ	transmission	μsec. per byte
ρ	data permutation	μsec. per byte
λ	startup (latency)	μsec.
δ	distance impact	μsec. per dimension

The time taken by a message of size m bytes to cross d dimensions is thus $\lambda + \tau m + \delta d$; the time to shuffle m bytes of data within memory is ρm. Expressions for the two algorithm are as follows.

In the Standard Exchange algorithm d transmissions of $m2^{d-1}$ bytes each over dimension 1, take $d(\lambda + \tau m 2^{d-1} + \delta)$ seconds. There are d shuffles on 2^d blocks of m bytes each, taking $d(\rho m 2^d)$ seconds. The total time is thus

$$t_s(m,d) = d(\lambda + (\tau + 2\rho)m2^{d-1} + \delta). \qquad (1)$$

In the case of the Optimal Circuit Switched algorithm there are $2^d - 1$ transmissions of blocks of m bytes. At each transmission step, all pairs of processors are at identical distances from each other. Thus the overall distance impact equals the average path length in a hypercube, which is $d2^{d-1}/(2^d - 1)$ The total time is

$$t_o(m,d) = (2^d - 1)(\lambda + \tau m + \delta \frac{d2^{d-1}}{2^d - 1}). \qquad (2)$$

The Standard Exchange algorithm is better than the Optimal Circuit Switched algorithm whenever

$$m < \frac{(2^d - d - 1)\lambda + d(2^{d-1} - 1)\delta}{(d2^{d-1} - 2^d + 1)\tau + d2^d\rho}.$$

For a hypothetical machine of dimension 6 with $\tau = \rho = 1, \lambda = 200$ and $\delta = 20$, the Standard Exchange algorithm is better for blocks of size less than 30.

5 The Multiphase Approach

We shall now describe a multiphase approach in which the complete exchange is carried out as a set of two or more "partial" exchanges. As we shall see, this permits us to use the Circuit Switched algorithm for block sizes for which it is ordinarily inefficient and provides very significant performance gains. In fact our multiphase approach is a unified algorithm that *includes* the Standard Exchange and Circuit Switched algorithms as special cases.

5.1 Motivation and Example

Given that the Standard Exchange algorithm is competitive for small message sizes and the Circuit Switched algorithm performs best at large message sizes, is there any way we can combine these algorithms to obtain performance better than either? This is indeed possible, as demonstrated below. Recall that we have $n = 2^d$ nodes on our hypercube. The normal complete exchange algorithm is based on the exchange of sets of n blocks per processor. We can envisage a "partial" exchange that is carried out simultaneously on all subcubes of dimension $d_1 < d$ but based on $n = 2^d$ blocks (not 2^{d_1} blocks) per processor. By carefully permuting our data blocks, we can then execute another partial exchange on all subcubes of dimension $d_2 = d - d_1$, again with 2^d blocks and not 2^{d_2} blocks. The end result will be that a complete exchange on the hypercube of dimension d is carried out in two phases, using messages that are longer than the messages that would have been used if a single phase approach had been employed. What we have achieved here is an effective "lengthening" of messages that lets us take advantage of the Circuit Switched algorithm for message sizes for which it is normally unsuited. The

000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
1								⇒			Partial Exchange, bits 2,1				
0:0	1:0	2:0	3:0	4:0	5:0	6:0	7:0	0:0	1:0	0:2	1:2	0:4	1:4	0:6	1:6
0:1	1:1	2:1	3:1	4:1	5:1	6:1	7:1	0:1	1:1	0:3	1:3	0:5	1:5	0:7	1:7
0:2	1:2	2:2	3:2	4:2	5:2	6:2	7:2	2:0	3:0	2:2	3:2	2:4	3:4	2:6	3:6
0:3	1:3	2:3	3:3	4:3	5:3	6:3	7:3	2:1	3:1	2:3	3:3	2:5	3:5	2:7	3:7
0:4	1:4	2:4	3:4	4:4	5:4	6:4	7:4	4:0	5:0	4:2	5:2	4:4	5:4	4:6	5:6
0:5	1:5	2:5	3:5	4:5	5:5	6:5	7:5	4:1	5:1	4:3	5:3	4:5	5:5	4:7	5:7
0:6	1:6	2:6	3:6	4:6	5:6	6:6	7:6	6:0	7:0	6:2	7:2	6:4	7:4	6:6	7:6
0:7	1:7	2:7	3:7	4:7	5:7	6:7	7:7	6:1	7:1	6:3	7:3	6:5	7:5	6:7	7:7
2								⇒				2-Shuffle			
0:0	1:0	0:2	1:2	0:4	1:4	0:6	1:6	0:0	1:0	0:2	1:2	0:4	1:4	0:6	1:6
0:1	1:1	0:3	1:3	0:5	1:5	0:7	1:7	2:0	3:0	2:2	3:2	2:4	3:4	2:6	3:6
2:0	3:0	2:2	3:2	2:4	3:4	2:6	3:6	4:0	5:0	4:2	5:2	4:4	5:4	4:6	5:6
2:1	3:1	2:3	3:3	2:5	3:5	2:7	3:7	6:0	7:0	6:2	7:2	6:4	7:4	6:6	7:6
4:0	5:0	4:2	5:2	4:4	5:4	4:6	5:6	0:1	1:1	0:3	1:3	0:5	1:5	0:7	1:7
4:1	5:1	4:3	5:3	4:5	5:5	4:7	5:7	2:1	3:1	2:3	3:3	2:5	3:5	2:7	3:7
6:0	7:0	6:2	7:2	6:4	7:4	6:6	7:6	4:1	5:1	4:3	5:3	4:5	5:5	4:7	5:7
6:1	7:1	6:3	7:3	6:5	7:5	6:7	7:7	6:1	7:1	6:3	7:3	6:5	7:5	6:7	7:7
3								⇒			Partial Exchange, bit 0				
0:0	1:0	0:2	1:2	0:4	1:4	0:6	1:6	0:0	0:1	0:2	0:3	0:4	0:5	0:6	0:7
2:0	3:0	2:2	3:2	2:4	3:4	2:6	3:6	2:0	2:1	2:2	2:3	2:4	2:5	2:6	2:7
4:0	5:0	4:2	5:2	4:4	5:4	4:6	5:6	4:0	4:1	4:2	4:3	4:4	4:5	4:6	4:7
6:0	7:0	6:2	7:2	6:4	7:4	6:6	7:6	6:0	6:1	6:2	6:3	6:4	6:5	6:6	6:7
0:1	1:1	0:3	1:3	0:5	1:5	0:7	1:7	1:0	1:1	1:2	1:3	1:4	1:5	1:6	1:7
2:1	3:1	2:3	3:3	2:5	3:5	2:7	3:7	3:0	3:1	3:2	3:3	3:4	3:5	3:6	3:7
4:1	5:1	4:3	5:3	4:5	5:5	4:7	5:7	5:0	5:1	5:2	5:3	5:4	5:5	5:6	5:7
6:1	7:1	6:3	7:3	6:5	7:5	6:7	7:7	7:0	7:1	7:2	7:3	7:4	7:5	7:6	7:7
4								⇒				1-Shuffle			
0:0	0:1	0:2	0:3	0:4	0:5	0:6	0:7	0:0	0:1	0:2	0:3	0:4	0:5	0:6	0:7
2:0	2:1	2:2	2:3	2:4	2:5	2:6	2:7	1:0	1:1	1:2	1:3	1:4	1:5	1:6	1:7
4:0	4:1	4:2	4:3	4:4	4:5	4:6	4:7	2:0	2:1	2:2	2:3	2:4	2:5	2:6	2:7
6:0	6:1	6:2	6:3	6:4	6:5	6:6	6:7	3:0	3:1	3:2	3:3	3:4	3:5	3:6	3:7
1:0	1:1	1:2	1:3	1:4	1:5	1:6	1:7	4:0	4:1	4:2	4:3	4:4	4:5	4:6	4:7
3:0	3:1	3:2	3:3	3:4	3:5	3:6	3:7	5:0	5:1	5:2	5:3	5:4	5:5	5:6	5:7
5:0	5:1	5:2	5:3	5:4	5:5	5:6	5:7	6:0	6:1	6:2	6:3	6:4	6:5	6:6	6:7
7:0	7:1	7:2	7:3	7:4	7:5	7:6	7:7	7:0	7:1	7:2	7:3	7:4	7:5	7:6	7:7

Figure 3: A Multiphase Exchange on a hypercube of dimension 3. The first row gives the binary labels of processors. Data blocks are arranged in columns. The first partial exchange is on the 2 subcubes of dimension 2 determined by bits 2 and 1; data are moved in superblocks of size 2. This is followed by a 2-shuffle. The second partial exchange is on the 4 subcubes of dimension 1 determined by bit 0; data are moved in superblocks of size 4.

price paid is the overhead of data permutation, which is required to align blocks to that they finish up in the correct position. Figure 3 illustrates this approach for a dimension 3 hypercube.

An example. Suppose we have to carry out the complete exchange of block size 24 on our hypothetical 6-dimensional hypercube (Section 4.3) with $\tau = \rho = 1, \lambda = 200$ and $\delta = 20$. We have seen that the Standard Exchange algorithm is best on this machine for blocksizes of less than 30 bytes. For 24 bytes the Standard algorithm takes 15144μsec. Let us see what happens if we carry out this exchange in two phases of dimension 2 and 4 respectively. The first phase on dimension 2 subcubes with an effective block size of $24 \times 2^{6-2} = 384$ bytes takes 1832μsec. using the Circuit Switched algorithm. The next exchange on dimension 4 subcubes with effective block size $24 \times 2^{6-4} = 160$ bytes takes 6040μsec., again using the Circuit Switched algorithm.

To this must be added the overhead of shuffling data, which is $\rho m 2^d$ per phase. This totals 3072μsec. The total time for the two phase approach is thus 10944μsec., which is substantially faster than the Standard algorithm.

5.2 General Algorithm

A complete exchange on a hypercube of dimension d with $n = 2^d$ processors and block size m is done using a set of partial exchanges $\mathcal{D} = \{d_1, d_2, \cdots, d_k\}$, where each d_i specifies a subcube dimension. Obviously $|\mathcal{D}| = k$, $1 \leq k$, and $\Sigma_{i=1}^{k} d_i = d$.

The jth partial exchange is done on the set of subcubes determined by bits $\Sigma_{i=1}^{j} d_i - d_j$ to $\Sigma_{i=1}^{j} d_i$ of the hypercube node labels.

In a partial exchange 2^d blocks of size m each are exchanged, regardless of cube dimension. Hence the time required for the ith phase is obtained from expression (1) or (2) with m replaced by $m2^{d-d_i}$ bytes. This is the *effective block size.* The multiphase algorithm is as follows.

```
procedure Multiphase;
{  $d$:      dimensicn of the hypercube
   $k$:      number of phases (subcubes) in partition $\mathcal{D}$
   $d_i$:    dimension of the $i$th subcube in partition $\mathcal{D}$
   $start$:starting bit of subcube label
   $stop$: ending bit of subcube label }
begin
   $start = d - 1$;
   for $i = 1$ to $k$ do
      {Partial exchange}
         begin
            $stop = start - d_i + 1$;
            compute effective blocksize;
            for $j = 1$ to $(2^{start-stop+1} - 1)$ do
               send_effective_block_to_processor($(mynumber) \oplus (j2^{stop})$);
            shuffle blocks $d_i$ times;
            $start = stop - 1$;
         end;
end;
```

When $k = d$, all d_is are 1. In this case the outer i loop is executed k times with $start = stop = d-1, d-2, \cdots, 1, 0$. The inner j loop is executed only once for each i. In this case Multiphase degenerates into Standard Exchange.

When $k = 1$ and thus $d_1 = d$, the outer loop is executed only once. $stop$ always equals 0 and, in the inner loop, j takes on the values $1, 2, \cdots, 2^d - 1$ and thus Multiphase becomes Optimal Circuit Switched.

6 Minimizing the Execution Time

The theory developed in Section 4 assures us that multiphase exchanges can be useful; the general algorithm of Section 5 tells us how the partial exchanges are to be performed. It remains to discuss the problem of determining the optimal set of subcube dimensions and algorithms.

Given a hypercube of dimension d, there are many different combinations of subcube dimensions and algorithms that can be used to obtain a multiphase algorithm.* The optimal set can be obtained by enumerating over all the partitions of d. For each partition $\mathcal{D} = \{d_1, d_2, \cdots, d_k\}$ we select the best algorithm at each phase. This procedure is not as expensive as it appears at first sight, since we are enumerating over the partitions of hypercube *dimension* and not *size*. It is a classical result [1, 6] that the number of partitions of an integer d is

$$p(d) \sim \frac{1}{4\sqrt{3}\,d} e^{\pi\sqrt{2/3}\sqrt{d}}.$$

Exact values can be calculated using the recurrence

$$p(d) = \sum_{j=1}^{(1+\sqrt{1+24d})/6} (-1)^{j+1} p(d - \frac{1}{2}j(3j \pm 1)).$$

The following table enumerates the values of practical interest. We can see that for a thousand node hypercube (the largest that was commercially available in 1990) we need to enumerate only 42 partitions—a trivial number. Even for a *million node* hypercube, the enumeration of 627 partitions is quite viable, especially since it needs to be done only once and the optimal combination stored for repeated future use.

p	$p(d)$	p	$p(d)$	p	$p(d)$	p	$p(d)$
1	1	6	11	11	56	16	231
2	2	7	15	12	77	17	297
3	3	8	22	13	101	18	385
4	5	9	30	14	135	19	490
5	7	10	42	15	176	20	627

7 Implementation on the iPSC-860

We have implemented the Multiphase algorithm on the Intel iPSC-860 hypercube. In this Section we discuss the salient features of our implementation and derive expressions for the predicted run times.

*The sequence of dimensions is unimportant, as long as the shuffles are carried out correctly.

7.1 Message Types

There are two message types (selectable by the programmer) on the iPSC-860[4]. A messages of the FORCED type is discarded upon arrival if no receive has been posted for it. A message of the UNFORCED type is stored in a system buffer if it arrives and no receive has been posted for it. The performance of both types is similar for messages of size 0–100 bytes. Beyond 100 bytes, an UNFORCED message is preceded by the exchange of 'reserve-acknowledge' messages that cause space to be reserved in the destination. This causes substantial overhead[2].

When the intercommunication pattern is fully known before runtime, as is the case for complete exchange, suitable receives can be posted at all processors before communication begins, and the more efficient FORCED type used. We have done so in our implementation.

7.2 Pairwise Synchronized Exchange

This issue arises because of an idiosyncrasy of the iPSC-860's communication hardware. A receive and a transmit occurring nearly simultaneously at a processor can proceed concurrently, while a short delay causes them to be carried out serially. This issue has been researched in detail by Seidel et al. [9, 13, 14]. They have shown that two processors can execute a pairwise exchange concurrently if the transmissions start simultaneously. This synchronization can be achieved by using a global synchronization before each exchange, but that is an extremely expensive solution.

It has been shown that a pairwise exchange is guaranteed to proceed concurrently if the two processors involved first exchange a pair of zero byte "pairwise synchronization" messages. The time for this pairwise synchronization is far less than the time for global synchronization and is negligible for moderate to large messages.

7.3 Global Synchronization

When using FORCED message types it is essential for each processor to post receives for all expected messages in the procedure at the very beginning, and to carry out a global synchronization after this. Omission of the (expensive) global synchronization step is fatal as it leads to messages arriving before their corresponding receives have been posted and thus being discarded by the operating system. When using UNFORCED messages, it is possible to omit this global synchronization step since *these messages are stored* by the operating system until the required receive has been posted. We have found that FORCED types give better performance, despite the overhead of global synchronization.

We use FORCED types for "pairwise synchronization" messages as well as for the actual data transfers. We post all receives for all messages before a global synchronization. This results in better performance than the method proposed in [13] which does not use global synchronization.

7.4 Measured Performance Characteristics

As discussed in Section 4.3, the time for a message of size m bytes to cross d dimensions is $\lambda + \tau m + \delta d$. When messages of the FORCED type are used and all receives are posted before transmission begins, the values λ and τ are 95.0μsec. and 0.394μsec./byte, respectively. The value of δ is 10.3μsec./dimension. The λ for a zero byte message is significantly better, being 82.5μsec. When using these measured parameters to predict the time required by the multiphase algorithm, we must remember that each pairwise exchange is preceded by an exchange of zero byte synchronization messages. Thus we have the *effective* values of $\lambda = 177.5\mu$sec. and $\delta = 20.6\mu$sec./dimension.

The time for global synchronization on a cube of dimension d has been measured at 150$d\mu$sec. The time for data permutation (shuffling) is $\rho = 0.54\mu$sec./byte. This is considerably slower than the time to transmit data because of the substantial overhead of computing the permutation. This occurs because we have implemented our algorithm in C using a compiler that does not take many of the powerful features of the iPSC-860 into account. It should be possible to significantly improve this figure by using assembly language and/or an optimizing compiler. This will change our final measured timings somewhat, but will not affect our overall approach, which is valid even if the cost of permutation is zero.

The time for a partial exchange on a subcube (Section 5.2) of dimension d_i within a hypercube of dimension d is thus

$$t_p(m, d_i, d) =$$
$$(2^{d_i-1} - 1)(177.5 + 0.394m + 20.6\frac{d_i 2^{d_i-1}}{2^{d_i} - 1} + 0.54 \cdot 2^d m) + 150d. \quad (3)$$

When $d_i = d$, the shuffling can be omitted altogether, since d-shuffles of 2^d blocks are equivalent to the identity permutation.

8 Evaluation of Multiphase Algorithm

We now present measured timings for the Multiphase algorithm on Intel iPSC-860 hypercubes of dimension 5,6 and 7. Our timings are presented as plots in Figures 4, 5 and 6 where we indicate each combination by its set of subcubes. Thus for dimension 5, the Standard Exchange algorithm is denoted by $\{1,1,1,1,1\}$ and the Optimal Circuit Switched Algorithm by $\{5\}$. For dimensions 5, 6 and 7, the number of combinations are 7, 11 and 15. Although we have measured the performance of all combinations, to avoid congested plots we show only those combinations that form the hull of optimality (i.e. only the best combination for every blocksize). The only exception is the Standard Exchange Algorithm ($\{1,1,\cdots\}$), which is shown for purposes of comparison, even though it is never optimal on the iPSC-860 for dimensions 5–7. Dashed lines on our plots indicate predicted values and solid lines show actual measurements.

As is to be expected, the Optimal Circuit Switched algorithm is always optimal for large enough block size. When $d = 5$ (Figure 4) the combination $\{2,3\}$ is optimal for block sizes less than 100 bytes. For $d = 6$, three combinations are optimal: $\{2,2,2\}, \{3,3\}$ and $\{6\}$. The last of these is optimal for message sizes beyond about 140 bytes. The first is optimal only for extremely small sizes. Figure 6 shows the plots for the largest iPSC-860 available ($d = 7$). In this case we again have three optimal combinations $\{2,2,3\}, \{3,4\}$ and $\{7\}$, with $\{7\}$ optimal beyond 160 bytes and $\{2,2,3\}$ optimal for 0 to 12 bytes. For $d = 7$, the combination $\{3,4\}$ leads to a factor of two improvement over both the Standard Exchange and Optimal Circuit Switched Algorithms at blocks of 40 bytes.

In all cases there is good agreement between the predicted and observed run times. However the agreement is not perfect, since the performance characteristics of the real iPSC-860 are much more complex than this simple model. Nevertheless our model is good enough to provide us with algorithms that can lead to substantial measured improvement that is of great practical relevance, given the ubiquity of the complete exchange pattern.

9 Conclusions

Circuit switched machines have only recently made an appearance as commercial products. These machines provide powerful communication mechanisms but, as the results of this paper show, very careful algorithm design is required to optimize performance.

We have addressed the problem of implementing the complete exchange (all-to-all personalized) pattern and have described a multiphase algorithm that unifies the two previously known algorithms and yields performance better than either over some ranges of message sizes. Similar techniques can be applied to other communication patterns. In particular, it will be interesting to see how the performance of the all-to-all broadcast, one-to-all personalized and one-to-all broadcast patterns[8] can be improved. Since the Complete Exchange is the most demanding *communication pattern*, the *time* taken by our multiphase algorithm is an upper bound on the time required by any of these patterns, in fact of *any* communication requirement. However it is challenging to exploit the structure of the simpler patterns so as to obtain even better performance.

An open theoretical issue is whether we can develop an efficient multiphase algorithm for a given arbitrary communication requirement (i.e. an arbitrary directed graph). A practical issue of interest is to evaluate the performance of the multiphase approach on the Ncube-2 circuit switched hypercube.

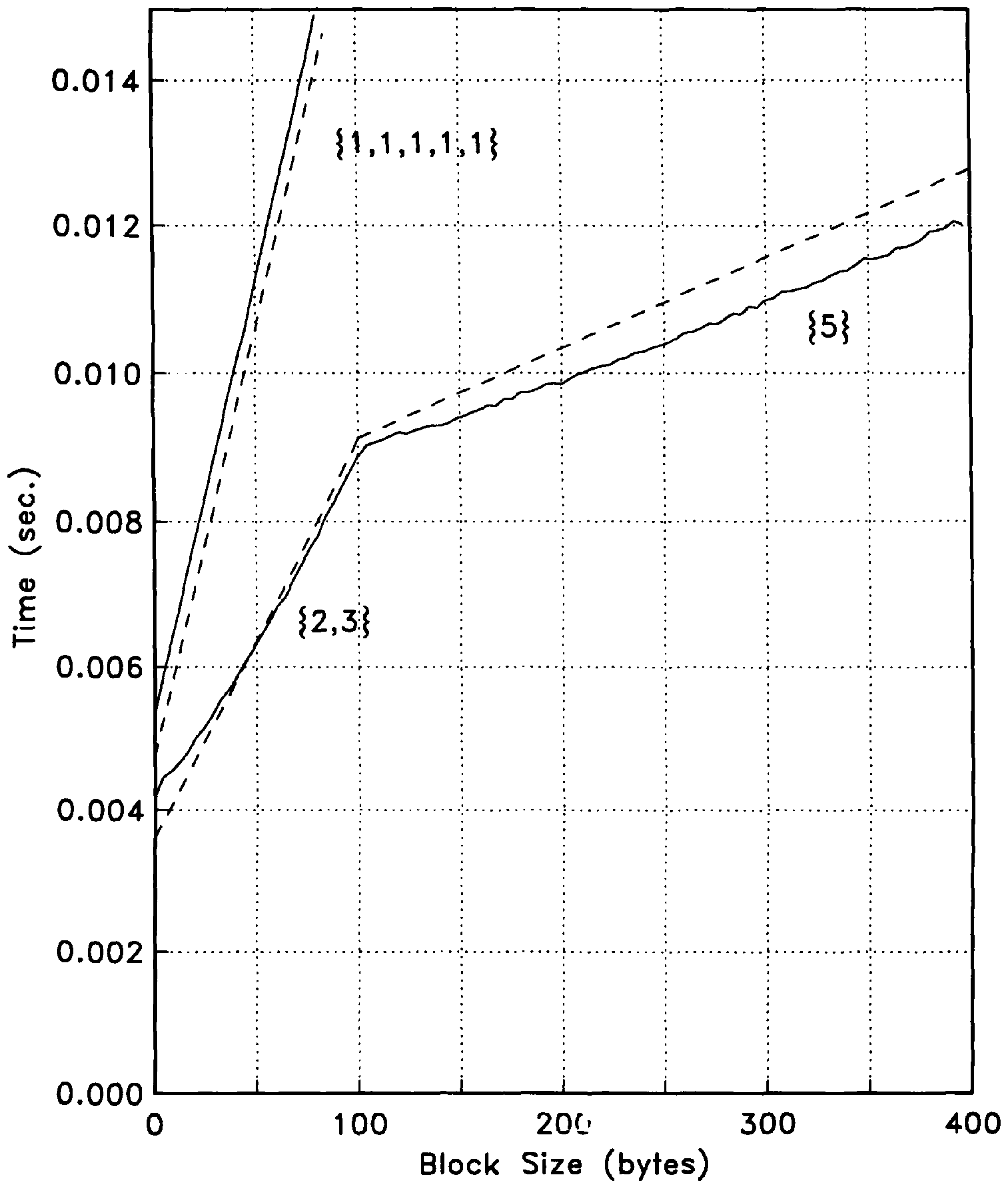

Figure 4: Performance of the Multiphase algorithm for a 32 node ($d = 5$) Intel iPSC-860. Solid lines indicate measured values; dashed lines are predictions. The hull of optimality is made up of two faces, corresponding to the partitions {2,3} and {5}. The Standard Exchange algorithm {1,1,1,1,1} is shown only for comparison–it is never optimal.

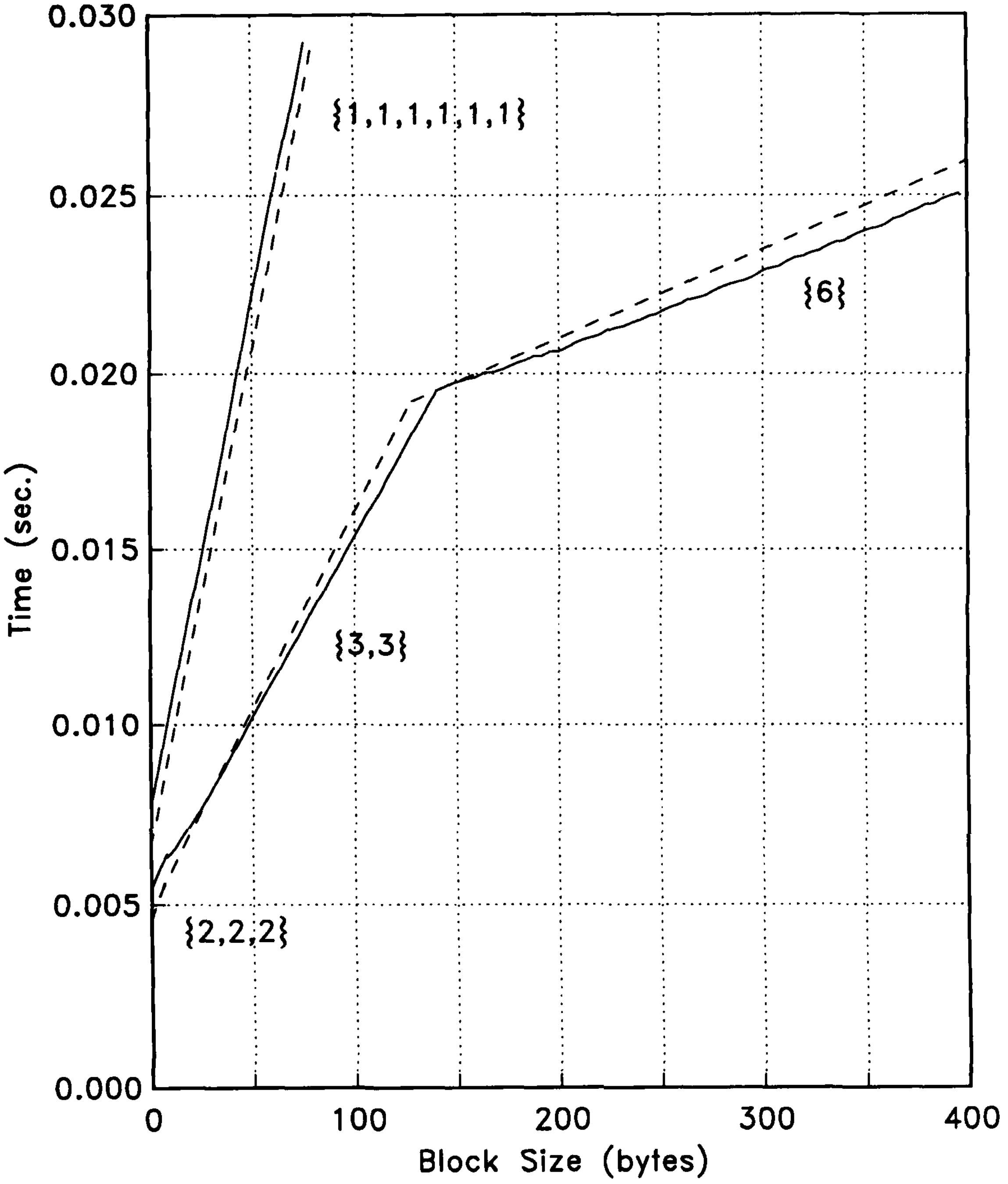

Figure 5: Performance of the Multiphase algorithm for a 64 node ($d = 6$) Intel iPSC-860. Three partitions are optimal in this case: {2,2,2}, {3,3} and {6}.

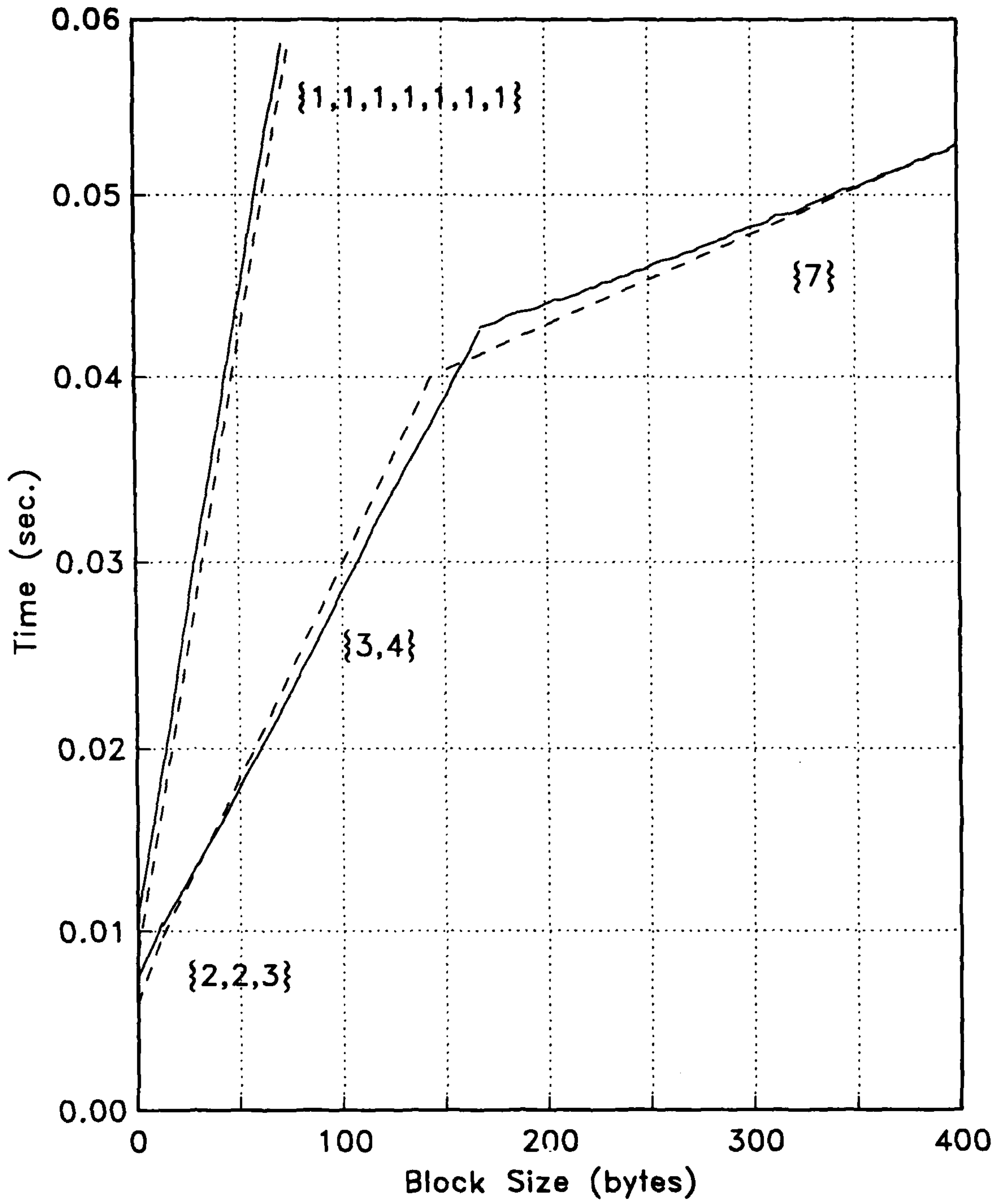

Figure 6: Performance of the Multiphase algorithm for a 128 node ($d = 7$) Intel iPSC-860. Three partitions are optimal in this case: {2,2,3}, {3,4} and {7}. For block size 40 bytes, the time taken by the Standard Algorithm {1,1,1,1,1,1,1} equals the time taken by the Optimal Circuit Switched Algorithm {6} and is 0.037 sec. The time taken by the Multiphase algorithm {3,4} is 0.016 sec., which is more than twice as fast.

Acknowledgements

I am indebted to John van Rosendale, Naomi Decker, Tom Crockett, Harry Jordan, David Nicol, Joel Saltz, Piyush Mehrotra, Ashraf Iqbal, Harry Berryman, Robert Voigt and Tom Zang for many useful discussions. I wish to thank Tom Crockett for getting me started on `bluecrab`, the 32 node iPSC-860 at ICASE. I am grateful to David Bailey for arranging access to `lagrange`, the 128 node iPSC-860 at NASA-Ames and to Leigh Ann Tanner for her help in using the machine.

References

[1] George E. Andrews. *The theory of partitions.* Addison-Wesley, Reading, Massachusetts, 1976.

[2] S. H. Bokhari. Communication overheads on the Intel iPSC-860 hypercube. ICASE Interim Report 10, May 1990.

[3] S. H. Bokhari. Complete exchange on the Intel iPSC-860 hypercube. Technical Report 91-4, ICASE, January 1991.

[4] Intel Corporation. iPSC/2 and iPSC/860 programmers reference manual, June 1990.

[5] J. Douglas and J. E. Gunn. A general formulation of alternating direction methods. *Numer. Math.*, 6(5), 1964.

[6] Emil Grosswald. *Topics from the Theory of numbers.* Birkhäuser, Boston, 1984.

[7] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988.

[8] S. Lennart Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, C-38(9):1249–1268, September 1989.

[9] Ming-Horng Lee and Steve R. Seidel. Concurrent communication on the Intel iPSC/2. Technical Report CS-TR 9003, Dept. of Computer Science, Michigan Tech. Univ., July 1990.

[10] D. W. Peaceman and H. H. Rachford. The numerical solution of parabolic and elliptic differential equations. *SIAM J.*, 3(1), 1955.

[11] Richard B. Pelz. The parallel Fourier pseudospectral method. *J. Comp. Phys*, 1990. To appear.

[12] J. Saltz, K. Crawley, R. Mirachandney, and H. Berryman. Run-time scheduling and execution of loops on message passing machines. *JPDC*, 8:303–312, 1990.

[13] Thomas Schmiermund and Steve R. Seidel. A communication model for the Intel iPSC/2. Technical Report CS-TR 9002, Dept. of Computer Science, Michigan Tech. Univ., April 1990.

[14] Steve Seidel, Ming-Horng Lee, and Shivi Fotedar. Concurrent bidirectional communication on the Intel iPSC/860 and iPSC/2. Technical Report CS-TR 9006, Dept. of Computer Science, Michigan Tech. Univ., November 1990.

[15] Steve R. Seidel. Circuit switched vs. store-and-forward solutions to symmetric communication problems. In *Proc. 4th. Conf. Hypercube Concurrent Computers and Applications*, pages 253–255, 1989.